Structured Data Types

- Unlike a simple data type which has only a single data item, a structured data type is one in which each value is a collection of component items.
- String is an example of a structured data type. Individual elements are accessed by index: `myString[3]`
- C++ has the following structured data types: `struct`, `union`, `class`, and arrays

Records (C++ structs)

- A record is a heterogeneous structured data type (`struct`).
- Each component of a record is called a field of the record (member), and each field is given a name called the field (member) name.

struct Syntax

```cpp
struct Syntax
StructDeclaration
    struct TypeName
    {
        MemberList
    }
MemberList
    DataType MemberName ;
    DataType MemberName ;
...
```

struct Example

```cpp
eNUM GradeType (A, B, C, D, F);
struct StudentRec
    {
        string firstName ;
        string lastName ;
        float gpa ;
        int programGrade ;
        int quizGrade ;
        int finalExam ;
        GradeType courseGrade ;
    };
StudentRec firstStudent ;
StudentRec student ;
int grade ;
```

Accessing Individual Components

- To access an individual member of a `struct` variable, you give the name of the variable, followed by a dot (period), and then the member name. This expression is called a member selector.
- Examples:
  - `firstStudent.gpa`
  - `student.finalExam`

Using structs

```cpp
cin >> student.finalExam ;
grade = student.finalExam + student.programGrade + student.quizGrade ;
if (grade >= 900)
    student.courseGrade = A ;
else if (grade >= 800)
    student.courseGrade = B ;
else if (grade >= 700)
    student.courseGrade = C ;
else if (grade >= 600)
    student.courseGrade = D ;
else
    student.courseGrade = F ;
```
Aggregate Operations on structs

- An aggregate operation is one that manipulates the struct as an entire unit.
- You can assign a variable of a struct type to another variable of that same type. You can pass a struct variable to a function and return it as a function’s value.
- You cannot input an entire struct variable with one statement. You cannot perform arithmetic or comparison operations on struct variables.

More About struct Declarations

- You can declare variable names within the struct declaration.
- Example:

```c++
struct StudentRec
{
    string firstName;
    string lastName;
};
```

One More struct Example

```c++
#include <iostream>
using namespace std;

struct player
{
    int assists;
    int points;
    int rebounds;
};

int triple_double (struct player);

int main()
{
    player shaq, the_admiral, kobe, jason;
    int shaq_td, jason_td;
    shaq.assists = 12;
    shaq.points = 23;
    shaq.rebounds = 10;
    jason.assists = 12;
    jason.points = 23;
    jason.rebounds = 10;
    shaq_td = triple_double(shaq);
    cout << "shaq's triple double is " << shaq_td << endl;
    jason_td = triple_double(jason);
    cout << "jason's triple double is " << jason_td << endl;
    return 0;
}
```

Hierarchical Records

- Records whose components are themselves records are called hierarchical records.
Hierarchical Record Example

```c
struct DateType
{
    int month;
    int day;
    int year;
};

struct StatisticsType
{
    float failRate;
    DateType lastServiced;
    int downDays;
};

struct MachineRec
{
    int idNumber;
    string description;
    StatisticsType history;
    DateType purchaseDate;
    float cost;
};
```

Accessing Hierarchical Record Members

- `machine.purchaseDate`
- `machine.purchaseDate.month`
- `machine.purchaseDate.year`
- `machine.history.lastServiced.year`

Data Abstraction

- The hierarchical description of the machine data is better than the flat one.
  - Elements are grouped together logically.
  - The date can be used again.
  - The details of the entities are pushed down to a lower level.

Abstract Data Types

- To cope with complexity, the human mind engages in abstraction—the act of separating the essential qualities of an idea or object from the details of how it works or is composed.
- To manage complexity, software developers regularly use two important abstraction techniques: control abstraction and data abstraction.
- An example of control abstraction is a function call. Ex. `4.6 + sqrt(x)`
- We use data abstraction when we define a new type. We concentrate initially on its logical properties and defer implementation details.

Categories of Abstract Data Type Operations

- In general, the basic operations associated with an abstract data type fall into three categories: constructors, transformers, and observers.
  - A constructor creates a new instance (variable) of an ADT.
  - A transformer builds a new value of the ADT, given one or more previous values of the type.
  - An observer allows us to observe the state of an ADT without changing it.